Nikke Tapanainen

# Datansyötön yksinkertaistaminen Lightning Component -käyttöliittymäkehyksen avulla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikkan koulutusohjelma

Insinöörityö

30.5.2018

Metropolia

| | |
|---|---|
| Tekijä(t)<br>Otsikko<br><br>Sivumäärä<br>Aika | Nikke Tapanainen<br>Datansyötön yksinkertaistaminen Lightning Component -käyttöliittymäkehyksen avulla<br><br>26 sivua<br>30.5.2018 |
| Tutkinto | Insinööri (AMK) |
| Tutkinto-ohjelma | Tietotekniikkan koulutusohjelma |
| Ammatillinen pääaine | Ohjelmistotekniikka |
| Ohjaaja(t) | Lehtori Outi Grotenfelt<br>Lehtori Simo Silander |


Tämä insinöörityö käsittelee asiakkaan sisäiseen käyttöön tulevaa, ohjattuun tiedon syöttöön tarkoitettua ohjelmaa ja sen tekoprosessia. Ohjelmaa käytetään asiakkaan tapauksessa tilaukseen ja laskutukseen tarvittavan asiakastiedon tallentamiseen Salesforce-alustalla.

Perinteisesti Salesforceen tiedonsyöttö tapahtuu yksi tietue kerrallaan ja vaatii sen, että käyttäjä muistaa luoda jokaisen tietueen erikseen. Tuotetussa ohjelmassa sen sijaan ohjataan käyttäjää askel kerrallaan haluttuun lopputulokseen, eli siihen, että kaikki tarvittava tieto on syötetty Salesforceen ohjelman käyttökerran lopuksi.

Insinöörityössä käsitelään Salesforcen Lightning Component -käyttöliittymäkehystä, jolla tuotettu ohjelma on pääosin tehty. Lisäksi käydään läpi Salesforcen perustoiminnallisuuksia ja yleistä tiedon hallintaa.

Työssä toteutettu kokonaisuus on mukautettavissa ilman koodin muokkausta, vaikkakin asetukset ovat suhteellisen monimutkaiset. Ylipäätänsä ohjelma täyttää sille annetut vaatimukset.



| | |
|---|---|
| Avainsanat | CRM, Pilvipalvelut, Salesforce |

| | |
|---|---|
| Author(s)<br>Title<br><br>Number of Pages<br>Date | Nikke Tapanainen<br>Simplification of Data Entry with the Lightning Component Framework<br><br>26 pages<br>30 May 2018 |
| Degree | Bachelor of Engineering |
| Degree Programme | Information Technology |
| Professional Major | Software Engineering |
| Instructor(s) | Outi Grotenfelt, Senior Lecturer<br>Simo Silander, Senior Lecturer |

This thesis is about the development of a guided wizard- type application, that is for the customers internal users. The initial use-case this application was created for, is data entry of a customers order process, including the required invoicing data. The application works on the Salesforce platform and was created using the tools provided by Salesforce.

By default records in Salesforce are created one by one and if any related records have to be created, the user will have to remember the correct sequence of records. To help avoid user-error and make this process faster over-all, the application was created. In the application the records are created in a guided manner, in predefined order. At the end of the application use event, all required data will have been filled in.

In this thesis I will go into the details of the Lightning Component Framework, which the application was mostly developed using. Salesforce basic functionality and data structure is also explained.

| | |
|---|---|
| Keywords | CRM, Cloud Services, Salesforce |

## Sisällys

## Lyhenteet

| | |
|---|---|
| **API** | Application Programming Interface, ohjelmointirajapinnassa kuvataan tapoja joilla toinen ohjelma voi käyttää toisen palveluita. |
| **CRM** | Customer Relation Management, asiakkuudenhallintajärjestelmäohjelmisto, jossa säilytetään ja hallitaan tietoja asiakkaista. |
| **DML** | Data Manipulation Language, Salesforcen tiedon tallennus- ja muokkauskieli. |
| **HTML** | HyperText Markup Language, webohjelmoinnissa yleisesti käytetty kieli. |
| **SFDC** | Salesforce.com, yleisesti käytetty nimitys Salesforce-alustasta. |
| **SOQL** | Salesforce Object Query Language, Salesforcen oma tiedon hakukieli. |
| **SQL** | Structured Query Language, tietokannasta tiedon hakemiseen tehty kieli. |

# 1 Johdanto

Salesforce tuotteena on pilvipalveluna toteutettu asiakkuudenhallintajärjestelmä (CRM). Tämän insinöörityön aihe syntyi asiakkaan tarpeesta saada tilauksien kirjaamisesta Salesforce.com-alustalla helpompaa ja nopeampaa verrattuna siihen, mitä vakionäkymä tarjoaa. Alkuperäinen suunnitelma oli tehdä yhden sivun applikaatio, jossa olisi kaikki tarvittava tieto heti täytettävissä. Suunnittelussa päädyttiinkin ratkaisuun, jossa tiedot täytettäisiin tietue kerrallaan toinen toisensa perään ja tietyssä järjestyksessä.

Työssä käydään läpi toteutuksessa käytettyjä työkaluja, kuten ohjelmointikieliä ja menetelmiä, Salesforce-ympäristön perustoiminnalliisuuksia ja itse lopputuloksena syntyneen ohjelmiston teknisiä ominaisuuksia. Isoimmassa osassa on Lightning Component Framework, jolla ohjelmiston käyttöliitymä on toteutettu.

Salesforcen Apex-kielestä käsitellään enimmäkseen objektien metatietojen käsittelyssä käytettyä Schema-nimiavaruutta, joka on merkittävässä osassa toteutetussa ohjelmassa. Tiedon tallentamiseen ja hakuun käytettyjä menetelmiä käydään myös lyhyesti läpi.

Työssä toteutin käyttöliittymän suunnittelun, sen ohjelmoinnin, asetusten ja back-end-osuuden toteututuksen. Osassa käyttöliittymää on käytetty joko Lightning Component -käyttöliittymäkehyksen vakiokomponenttejä tai vapaasti saatavilla olevia Lightning-komponentteja.

# 2 Salesforce

Salesforce on Salesforce.com inc.:n omistama täysin pilvipalveluna toteutettu asiakkuudenhallintajärjestelmä. Pilvipalveluilla on tavanomaisesti rajoituksia koodin suoritukseen littyen, Salesforce ei ole poikkeus, muun muassa yhden koodisuorituskerran käyttämä prosessointiaika ja datan haku, ja muokkauskutsujen määrä, ovat rajoitettuja [1]. Salesforce.com-instansseista puhutaan organisaatioina ja yleinen lyhenne Salesforce.com-alustalle on SFDC.

## 2.1 Tietorakenne

Salesforcen datamalli rakentuu objektimääritelmistä, joissa on kenttiä ja muita tietoja. Yksi objekti on tiettyyn tarkoitukseen käytettävä tietomalli. Objektin ilmetymää kutsutaan tietueeksi. Objekti on verrattavissa yksittäiseen SQL-tyylisen tietokannan tauluun, ja tietue on sen taulun yksittäinen rivi [2].

### 2.1.1 Objektit

Objekteja on Salesforcessa valmiina ja niitä kutsutaan vakio-objekteiksi. Näille vakio-objekteille voidaan lisätä uusia kenttiä ja niitä voidaan poistaa, mutta objektilla olevia vakiokenttiä ei voida poistaa. Myöskään koko vakio-objektia ei voida poistaa.

Uusia objekteja voidaan luoda ja niitä kutsutaan mukautetuiksi objekteiksi. Mukautettuja objekteja voidaan muokata vapaammin kuin vakio-objekteja. Ne voi esimerkiksi poistaa kokonaan. Mukautetuille objekteille tulee luonnin yhteydessä muutama vakiokenttä, muun muassa nimi, luontiaika, omistaja ja viimeisin muokkausaika.

Kullakin objektilla on käyttöliittymässä näytettävän nimen lisäksi uniikki rajapintanimi (eng. API name), jota käytetään koodissa. Rajapintanimi vakio-objekteille on useimmiten sama kuin selkokielinen käyttöliittymässä näkyvä nimi, kun taas mukautetuilla

objekteilla rajapintanimen perässä on aina __c pääte. Koodissa Salesforcen objektit implementoivat sObject-luokkaa.

Objekteille voidaan määrittää eri käyttöihin tulevia sivuasetteluja (eng. Page Layout), joissa on vain tietyt objektin kentät näkyvissä. Näillä asetteluilla saadaan aikaan uudelleenkäytettävyyttä objekteille, jos esimerkiksi yhtä objektia laajentamalla ja asettelulla vain tarvittavan tiedon näyttäminen riittää uuden toiminnallisuuden toteutukseen.

Objektikohtaisien tietuetyyppien (eng. Recordtype) avulla voidaan erotella objekteille useita käyttötarkoituksia. Tietuetyypeillä määritellään, mitä sivuasettelua tullaan käyttämään ja mitkä valikko-tyyppisten arvot ovat valittavissa kyseisellä tyypillä. Jos objetille on määritetty useita tietuetyyppiä, tulee käyttäjän uutta tietuetta luodessa valita sille tyyppi. Koodissa tietuetyyppejä käytettäessä tietueelle tulee asettaa tietuetyypin tunniste eli id.

Objekteille voidaan luoda myös kenttäjoukkoja, joita voidaan käyttää koodissa käyttöliittymän luonnin yhteydessä. Kenttäjoukot ovat oikeastaan vain listoja kentistä, jossa kentille voidaan asettaa tieto siitä, onko kyseinen kenttä vaadittu tallennuksen yhteydessä vai ei.

Tiedon oikeellisuutta voidaan hallita objektikohtaisesti vahvistussäännöillä, jotka estävät tiedon tallentamisen, mikäli se ei vastaa määriteltyä muotoa tai muuta sääntöä [3]. Vahvistussäännöt ovat muodoltaan kaavoja, joiden tulos on tyyppiä totuusarvo. Jos kaava käy toteen, estyy tietueen tallentaminen kokonaan, ja käyttäjä saa vahvistussäännön mukaisen virheilmoituksen.

### 2.1.2 Kentät

Objekteille voidaan lisätä monentyyppisiä kenttiä kuten teksti, numero, viite toiseen objektiin. Kuten mukautettujen objektien kanssa, myös mukautetut kentät saavat erillisen rajapintanimen, jota tulee käyttää koodissa. Tämä rajapintanimi päättyy myös merkkeihin __c, kuten mukautettujen objektien kohdalla.

## 2.2 Käyttöliittymä

Salesforcessa on valmiiksi käyttöliittymä, jota on mahdollista laajentaa ja muokata haluamansa mukaan, tiettyjen rajojen sisällä. Käyttäjille on saatavilla joko vanhempi Classic-näkymä tai uusi Lightning Experience -näkymä.

Välilehti (eng. Tab) Salesforcessa voi sisältää yhden tyyppisen objektin tai mukautetun sovelluksen. Jokaisella objektilla on tyypillisesti oma välilehti, Visualforce-sivulle tai Lightning-komponentille voidaan myös erikseen luoda oma välilehti.

## 2.3 Mukautettu metatieto

Mukautettu metatieto (eng. Custom metadata) -tyypit ovat niin sanotusti meta-metatietoa ja niiden tietueet ovat metatietoa [4]. Mukautettuja metatietotyyppejä ja niiden tietueita voidaan siirtää Salesforce-ympäristöstä toiseen, esimerkiksi koodin mukana, mikä helpottaa sovelluksien paketointia. Toisin kuin mukautettujen asetuksien kanssa, joiden objektimääritelmät siirretään koodin tavoin ja itse asetustietueet tulee siirtää erikseen datasiirrolla.

Pääasiallinen tarkoitus mukautetulle metatiedolle on mahdollistaa ohjelmakokonaisuuden mukauttaminen ilman, että koodia tarvitsee muokata. Muun muassa ympäristöstä riippuvia asetuksia on monesti määritelty mukautettuun metatietoon, hallinan helpottamisen vuoksi.

Mukautetun metatiedon tietorakenne muistuttaa mukauettujen objektien tietorakennetta, niitä vain luodaan ja muokataan, joko Metadata-rajapintaa käyttäen tai Salesforcen Asetukset-osiossa [5].

# 3 Ohjelmistokehitys Salesforce-alustalla

## 3.1 Apex

Apex-kieli on yksi tapa toteuttaa omaa logiikkaa Salesforce-ympäristössä. Apex muistuttaa kieliasultaan Java- ja Microsoftin C# -ohjelmointikieliä. Back-end ja niin sanotusti ”luukun alla”oleva koodi Salesforcessa on Apexia. Apexilla voidaan luoda muun muassa SOAP- ja REST-rajapintoja, tiedon tallentamisen yhteydessä ajettavia Apex-käynnistimiä ja ajastettuja toimintoja.

### 3.1.1 Schema

Apexin Schema-nimiavaruus sisältää luokkia ja metodeita, joilla voidaan hakea koodissa Salesforcen objektien ja niiden kenttien metatietoja [6]. Näiden metatietojen perusteella voidaan esimerkiksi simuloida Salesforcen vakionäkymää.

Schema-nimiavaruus mahdollistaa hyvin dynaamisen lähestymistavan käsitellä organisaation datamallia, sillä haettavan metatiedon tyyppejä voidaan käsitellä merkkijonoina Schema-metodia käytettäessä [7]. Tämä mahdollistaa sen, että haettavan metatiedon nimiä ei tarvitse tietää ohjelmaa kirjoittaessa, vaan ne voidaan käytön mukaisesti määrittää. Objekteja ja kenttiä käsitellään token-tyyppisten oloiden kautta, jotta muistin käyttö olisi pienempää.

DescribeSObjectResult-luokan avulla saadaan yksittäisestä Salesforcen objektista hyvin paljon tietoa selville, kuten esimerkiksi lista sen kaikista kentistä. Tämän luokan avulla voidaan myös tarkistaa, minkä tasoiset oikeudet käyttäjällä on kyseiseen objektiin ja sen kenttiin.

### 3.1.2 DML

Data Manipulation Language (DML) on tapa muokata tietoa Salesforce-ympäristössä [8]. Tietueita voidaan tallentaa joko yksi kerrallaan tai joukkossa, joista jälkimmäinen on suositeltavampi tapa. Sillä ohjelman suorituskertakohtaisten rajoitusten mukaan 150 on maksimimäärä yksittäisiä DML-operaatioita, kun taas yksittäisen tietueen transaktioita voidaan suorittaa 10 000.

DML-kutsuja käytetään Apex-koodissa yksinkertaisilla insert-, update-, upsert- tai delete-komennoilla. Apexissa on käytettävissä myös Database-luokka, jonka metodeilla saadaan aikaan sama lopputulos kuin DML-komennoilla, mutta lopputulosta voidaan operaation jälkeen tutkia koodissa [9]. Database-luokka mahdollistaa myös sen, että vaikka osa tallennettavista tietueista olisikin virheellinen, niin muut voidaan tallentaa onnistuneesti, kun taas DML-komennoilla tämä ei onnistu.

```
1 Account acc = new Account(Name = 'Test Account');
2 insert acc;
3 acc.Email = 'test.mail@example.com';
4 update acc;
```

Esimerkkikoodi 1: Esimerkki DML-kutsujen käytöstä.

Tallentaminen voi keskeytyä virhetilanteeseen, mikäli jokin tallennettavasta tiedosta on jollakin tapaa virheellistä. Esimerkiksi numerokenttään voidaan olla yrittämässä tallentaa tekstiä tai mukautettujen vahvistussääntöjen vastaista tietoa ollaan tallentamassa. DML-kutsuja käytettäessä tallentamisen virheet aiheuttavat DML-poikkeuksen (eng. DML-Exeption), joka aihteuttaa koodin suorittamisen keskeytyksen samalla tavalla kuin mikä tahansa muukin Apex-poikkeus.

## 3.2 Visualforce

Visualforce on samantapainen merkintäkieli kuin tunnetuin tapa toteuttaa nettisivuja, HyperText Markup Language (HTML). Se tukee samoja merkintöjä ja noudattaa samoja sääntöjä kuin HTML. Visualforcessa on käytettävissä lukuisia lisäominaisuuksia perus-HTML:lään verrattuna, kuten apex-nimiavaruuden tarjoamat, valmiiksi tyylitellyt komponentit ja helppo integraatio back-end-luokkiin.

Salesforce tarjoaa valmiita back-end-luokkia objekteille. Näitä kutsutaan standardi-ohjaimiksi (eng. Standard Controller) Visualforce-kontekstissa. Näiden ohjainluokkien avulla voidaan käyttää nykyisen objektin tietuetta, kun Visualforce-sivu on upotettuna objektin tietuesivulla.

## 3.3 Lightning Component Framework

Lightning Component Framework on käyttöliittymäkoodiohjemistokehikko, joka sisältää valmiiksi monia peruskomponentteja ja mahdollistaa uusien, täysin mukautettujen applikaatioiden ja komponenttien luonnin. Lightning-komponentit ovat itsenäisiä ja uudelleen käytettäviä Lightning-applikaation osia [10]. Lightning-applikaatiot ja -komponentit toimivat vain Lightning Experience -näkymässä, ellei niitä sisällytetä Visualforce-sivulla Lightning out-toiminnallisuutta hyödyntäen.

### 3.3.1 Lightning-applikaatiot

Lightning-applikaatiot (eng. Lightning App) ovat johonkin tiettyyn tarkoitukseen tehtyjä kokonaisuuksia, jotka usein koostuvat monesta Lightning-komponentista. Rakenteeltaan lightning-applikaatiot ovat hyvin samanlaisia kuin Lightning-komponentit, applikaatiolla ollessa muutama järjestelmäattribuutti enemmän.

### 3.3.2 Lightning-komponentit

Lightning-komponentit koostuvat useimmiten HTML-pohjaisesta näkymästä, Javascript-ohjaimesta, JavaScript-apuluokasta ja CSS-tyylimääritelmästä [10]. Näkymäosio sisältää määritelmät attribuuteille, tapahtumille, metodeille ja itse HTML-koodin, jonka mukaan komponentin näkyvää osuus rakentuu. HTML-koodin sekaan voidaan myös käyttää Aura-nimiavaruuden merkintöjä, kuten aura:iteration-merkintää, joka on yksinkertainen toistorakenne. Muita komponentteja voidaan myös sisällyttää HTML-koodin seassa, jolloin niiden oma näkymäosio tulee siihen kohtaan näkyviin.

Komponentin elinkaaren aikaiset muuttujatyyppiset attribuutit määritellään sen näkymässä, HTML-merkinnän omaisesti. Näihin attribuutteihin voidaan viitata HTML-koodin seassa ja komponentin Javascript-koodissa. Attribuuttejen arvoja voidaan muokata lennosta, ja ne päivittyvät samalla paikoissa, joissa on viittaus.

Lightning-komponenteissa toiminnallisuus kirjoitetaan JavaScriptillä erillisissä ohjain- ja apuluokisssa. Ohjainluokan funktioita voidaan kutsua näkymäosiosta, esimerkiksi nappulan onclick-määritteessä. Ohjainluokka käyttää apuluokan funktioita ja hyvänä kehitystapana on kirjoittaa yksinkertaisia ja uudelleenkäytettäviä funktioita apuluokkaan ja käyttää niitä ohjainluokassa tarpeen mukaan.

Lightning-komponentit voivat sisältää toisia Lightning-komponentteja ja voivat jakaa keskenään muunmuassa attribuuttiviittauksia. Komponentin attribuuteille ja metodeille voidaan määrittää näkyvyystaso [11], joka määrää kyseisen attribuutin tai metodin näkyvyyden komponentin ulkopuolelle. Näitä näkyvyystasoja on käytettävissä kolme erilaista, yksityinen (eng. private), joka asettaa kohteen näkyväksi vain kyseisen komponentin sisälle. Taso julkinen (eng. public) asettaa kohteen näkyväksi kaikille muille komponenteille ja applikaatioille saman organisaation sisällä. Tämä on oletustaso, eli jos näkyvyystasoa ei erikseen määritellä, tulee tämä käyttöön. Viimeinen kolmesta tasosta on maailmanlaajuinen (eng. global), joka on muuten sama kuin julkinen, mutta kohteet ovat näkyvissä myös kohdeorganisaation komponenteille, jos komponentit paketoidaan ja asenettaan toiseen organisaatioon.

Lightning-komponenttien toteutus on kapseloitu, kuten esimerkiksi olio-ohjelmoinnista tutulla luokka- ja oliomenetelmällä. Komponentin sisällyttäminen toisessa toimii hyvin samalla tavalla kuin uuden olion luominen, ja uusia komponenttien ilmentymiä voidaan sisällyttää emokomponentissa useampia. Komponentit ovat toisistaan riippuumattomia, ellei niissä käytetä samoja emokomponentin arvoviittauksia tai olla rakennettu tapahtumilla riippuvuussuhteita.

```
1 <aura:component controller="EsimerkkiApexOhjain">
2   <aura:attribute name="nimi" type="String"/>
3   <aura:attribute name="tervehdys" type="String" default="Hei!"/>
4
```

```
5     <ui:inputText label="Nimi" value="{!v.nimi}"/>
6     <p>{!v.tervehdys} {!v.nimi}</p>
7   </aura:component>
```

Esimerkkikoodi 2: Yksinkertainen Lightning-komponentti, joka mahdollistaa käyttäjän antamaan nimen ja näyttää sen alla tervehdystekstin, jossa syötetty nimi on.

### 3.3.3 Lightning-tapahtumat

Komponenttien välinen kommunikaatio voidaan hoitaa Lightning-tapahtumilla (eng. Lightning Event). Tapahtumia on käytettävissä kahta erilaista: joko komponentti- tai applikaatiotason [12]. Komponenttitason tapahtumat hallitaan joko itse komponentissa tai sen emokomponentissa. Applikaatiotason tapahtuma voidaan vastaanottaa jokaisessa komponentissa, joka sitä kuuntelee.

Yksittäiselle lightning-tapahtumalle on luotava oma metatieto Salesforceen. Tapahtuman koodissa määritellään attribuutit, joiden arvot kulkevat tapahtuman mukana. Attribuuttit määritellään samalla tavalla kuin lightning-komponentin näkymäosiossa [12]. Koodiesimerkissä 3 luodaan yksinkertainen Lightning-tapahtuma nimeltä kmpTapahtuma, joka on komponenttitason tapahtuma ja sillä on yksi merkkijonotyyppinen atribuutti, nimeltään ”viesti”.

```
1   <!--c:kmpTapahtuma-->
2   <aura:event type="COMPONENT">
3       <aura:attribute name="viesti" type="String"/>
4   </aura:event>
```

Esimerkkikoodi 3: Lightning-tapahtuma, jossa yksi atribuutti

Lightning-komponentissa merkitään tapahtumat, joita se kuuntelee ja myös komponenttitason tapahtumat, joita komponentti laukaisee.

```
1   <aura:registerEvent name="esimerkkiTapahtuma" type="c:kmpTapahtuma"/>
2   <aura:handler name="esimerkkiTapahtuma" event="c:kmpTapahtuma"
        action="{!c.handleEvent}"/>
```

Esimerkkikoodi 4: Tapahtuman rekisteröinti ja kuuntelijan määritys

Koodiesimerkissä 4 ensin rekisteröidään aikaisemmassa esimerkissä luodun tapahtuman ilmentymä ja sille annetaan nimeksi ”esimerkkiTapahtuma”. Esimerkin toisella rivillä määritetään tälle tapahtumalle kuuntelija, jonka nimi tulee olla sama kuin tapahtuman

ilmentymällä [13].

Tapahumat laukaistaan komponentin JavaScript-osiosta, jossa asetetaan samalla tapahtuman mahdolliset parametrit.

```
1  fireEvent : function(component, event){
2    var event = component.get("esimerkkiTapahtuma");
3    event.setParams({
4      "viesti" : "Hello world!"
5    });
6    event.fire();
7  },
```

Esimerkkikoodi 5: Tapahtuman atribuutin arvon asettaminen ja itse tapahtuman laukaiseminen

Koodiesimerkissä 5 on JavaScript-funtio, joka ensin hakee komponentin merkinnästä tapahtuman nimeltä ”esimerkkiTapahtuma”, asettaa sen ”viesti- parametrille arvon ”Hello world!”ja lopulta laukaisee sen.

```
1  handleEvent : function(component, event){
2    var eventViesti = event.getParam("viesti");
3  },
```

Esimerkkikoodi 6: Tapahtuman käsittely ja sen atribuutin tallentaminen paikallismuuttujaan

Koodiesimerkissä 6 on palapelin viimeinen palanen, eli funktio, joka suoritetaan, kun tapahtuman kuuntelija vastaanottaa tapahtuman. Tämä on sama funktio, joka määritellään komponentin näkymässä registerEvent-merkinnässä action-parametriksi.

### 3.3.4 Tiedon siirto palvelimen ja käyttöliittymän väillä

Lightning-komponentit pystyvät käyttämään palvelimella olevia Apex-luokkien metodeita, jos kyseinen luokka on Lightning-komponentissa määritetty controller-attribuutissa ja luokan metodi on staattinen, merkitty @AuraEnabled-annotaatiolla [14] ja sen tulee olla luokan ulkopuolelle näkyvissä (public-näkyvyystaso tai laajempi).

Apex-metodeille käyttöliittymästä voidaan parametreinä toimittaa tietoa, tieto palvelimelta tulee tavanomasella return-komennolla. Muun muassa kokonaisia sObject-

tyyppisiä tietorakenteita voidaan käyttää siirtämään sellaisenaan palvelimelle ja takaisin.

```
1 @AuraEnabled
2 public static String sanoHei(String nimi){
3   return 'Hei ' + nimi + '!';
4 }
```

Esimerkkikoodi 7: Apex-metodi jota voidaan kutsua Lightning-komponentista, mikä ottaa vastaan merkkijonoargumentin ja palauttaa merkkijonon.

### 3.3.5 Salesforce Lightning Design System (SLDS)

Salesforce Lightning Design Systemin (SLDS) resurssien avulla voidaan luoda yhtenäistä ja Salesforce Lightningin mukaisia käyttöliittymiä [15]. SLDS sisältää muu nmuassa CSS-luokkia, SVG-ikoneita ja tyylitysohjeita. SLDS on valmiiksi sisäänrakennettuna Lightning-komponentteihin ja Visualforce-sivuissa sen saa käyttöön yhdellä tagilla.

SLDS on siitä hyödyllinen, että sitä ei tarvitse itse päivittää, kun Lightning Experience-käyttöliittymä esimerkiksi muuttuu. Salesforce itse ylläpitää SLDS:sää ja sen uusin versio on käytettävissä jokaisessa Salesforce-organisaatiossa. Kun käyttää komponenteissaan yksinomaisesti SLDS-tyyliluokkia, voi olla aika varma siitä, että sen ulkoasu pysyy muun standardin käyttöliittymän kanssa.

### 3.3.6 Lightning Out

Lightning out on tapa upottaa Lightning-komponentti esimerkiksi Visualforce-sivulle. Lightning out tarvitsee Lightning-aplikaation, joka laajentaa ltng:outApp:ia ja sisältää aura:dependency-attribuutit niille komponenteille, jotka muuten olisivat sen sisällä. Lightning Out Visualforce -sivun sisällä mahdollistaa käyttöliittymä-kehityksen tehtävän kokonaan Lightning-komponenteilla, vaikka olisikin tarve tehdä uutta myös Classic -näkymään.

### 3.4 Salesforce Object Query Language SOQL

Salesforce Object Query Language (SOQL) on Salesforcen oma SQL-tyylinen tiedon kyselykieli, joka on tarkoitettu nimenomaan Salesforce-datan hakemiseen [16]. SOQL-kyselyitä voidaaan suorittaa suoraan Apex-koodissa ja ympäristö osaa tyypittää palautetun tiedon valmiiksi muuttujaan sijoitettavaksi.

```
1 List<Account> accounts = [SELECT Id, Name FROM Account WHERE Name LIKE
      'Test%' LIMIT 2];
```

Esimerkkikoodi 8: Kysely Apex-koodissa, joka hakee Account-tietueita, joiden nimi alkaa merkkijonolla ”Test”.

## 4 Toteutus

Toteuteuttamaani sovellusta käytetään tilauksiin ja laskutukseen tarvittavan tiedon syöttöön, mutta sitä voidaan myös uudelleenkäyttää muunlaisen tiedon kanssa. Tieto syötetään objektikohtaisissa vaiheissa, joista yksi on kerrallaan näkyvissä.

Vaiheisiin perustuva käyttöliittymäsuunnitelma mahdollisti applikaation käyttöliittymän pienempään kokoon saamisen seka koko prosessin keskeyttämisen ilman, että kaikki jo täytetty tieto häviäisi. Tiedon oikeellisuus tulee myos tarkistettua joka kerta kun siirrytään seuraavan vaiheeseen. Toteutus muistuttaa hieman Microsoftin ohjelmistojen asennuksesta tuttua ohjattua toimintoa, jota myös velhoksi (eng. wizard) kutsutaan, jossa ohjelman käyttöönoton yhteydessä määritetään asetukset.

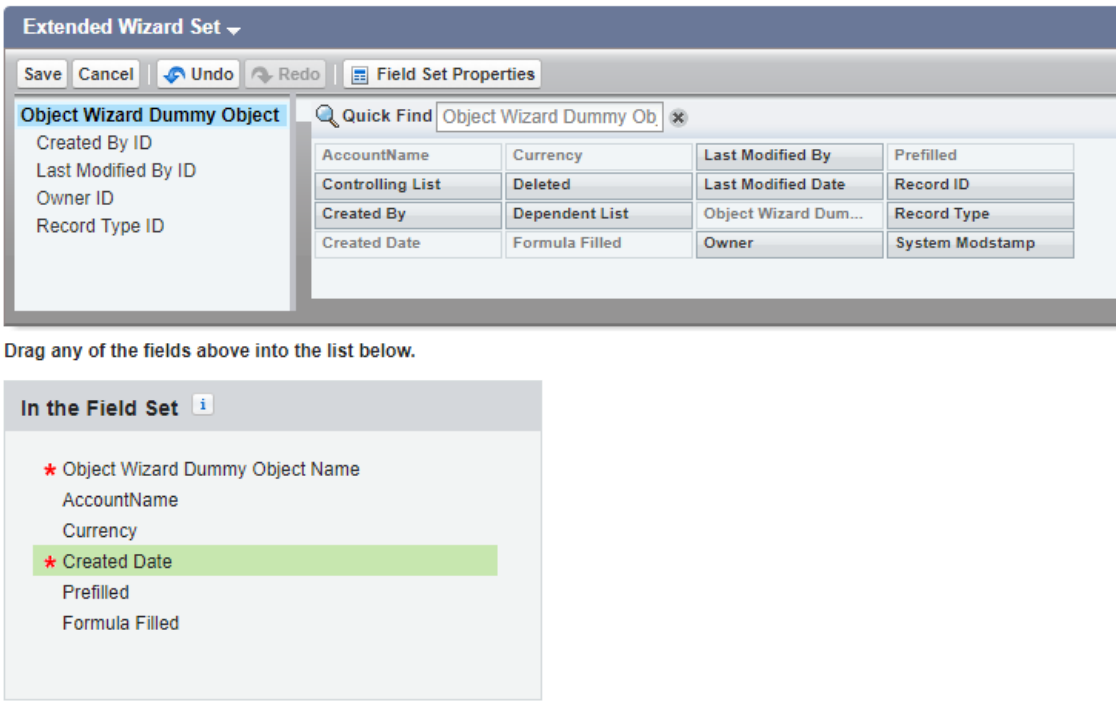
Yhtenä päävaatimuksena oli saada esitäyttettyä tietoa vaiheisiin, joko edeltä määritellyistä arvoista tai sitten käyttäjän aikaisemmin täyttämän tiedon perusteella. Tämä nopeuttaisi huomattavasti tiedonsyöttöä, kun samaa tai johdannaista tietoa ei tarvitse syöttää uudestaan tai turhaan. Myös käyttäjävirheet pienenevät.

Sovellus toteutettiin Salesforcen rajapintoja ja ohjelmistokehyksiä hyödyntäen. Tietueiden luonti tapahtuu vaiheittain asetusten mukaisessa järjestyksessä ja näkymissä. Kovakoodattuja toiminnallisuuksia ja arvoja pyrittiin välttämään sekä käyttöliittymä- että back end -koodissa.

## 4.1 Asetukset

Tuotetun sovelluksen näkyvä sisältö on muokattavissa mukautetun metadatan ja kenttäjoukkojen kautta. Tämä mahdollistaa sen, että vaiheiden muokkaamiseen ei tarvita koodin muuttamista.

### 4.1.1 Kenttäjoukko

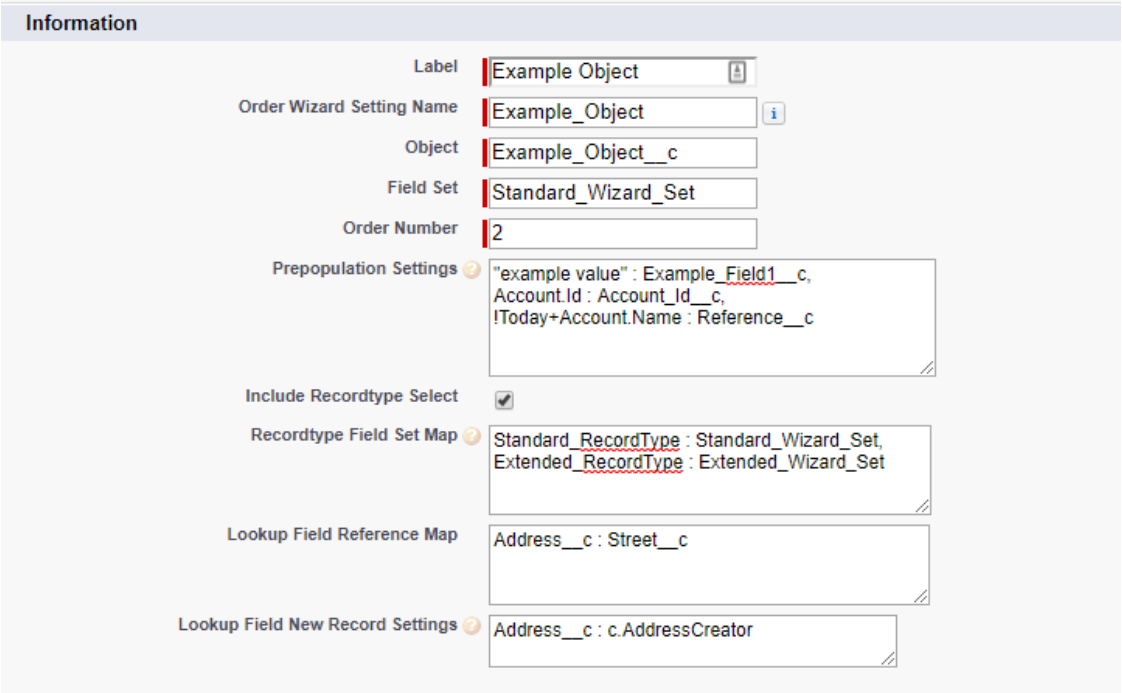



Kuva 1: Esimerkkikenttäjoukko

Kenttäjoukoissa listataan kentät, jotka halutaan näkyviin yksittäisessä vaiheessa. Myös se, mitkä kentät ovat vaadittuja asetetaan. Kenttien järjestys on sama sovelluksessa kuin kenttäjoukossa. Kuvassa 1 on sovelluksen testauksessa käytetyn mukautetun objektin kenttäjoukko.

### 4.1.2 Mukautettu metatieto

Sovellus ottaa vastaan mukautetun metatietotyypin nimen ja käy läpi jokaisen sen metadatan.




Kuva 2: Esimerkki mukautetusta metadatasta

Mukauettussa metatiedossa määritellään yksittäinen sovelluksen tietueen luontivaihe kuvan 2 mukaisesti. Object -kentässä asetetaan objekti, jonka ilmentymä tullaan luomaan. Field Set -kentässä on kenttäjoukko, jota oletusarvoisesti tullaan käyttämään. Order Number -kentässä määritellään kyseisen vaiheen järjestysnumero.

Prepopulation Settings -kentässä voidaan asettaa objektin kenttille oletusarvoja joko kovakoodattuna, viittauksena jonkin edellisessä vaiheessa luodun tietueen kenttään tai kaavalla. Erilliset sijoitukset erotellaan pilkulla ja jokaisessa ensin tulee sijoitettava arvo, kaksoispiste ja viimeisenä kohdekenttä. Kovakoodatut arvot kirjoitetaan lainausmerkkien sisään, viittaukset muiden vaiheiden arvoihin kirjoitetaan kyseisen vaiheen objektin nimi, piste ja haluttu kenttä. Kaavojen avulla voidaan sisällyttää nykyinen päivämäärä tai käyttäjän nimi, kovakoodatun tai viittauksen kanssa yhteen kenttään.

Include Recordtype Select -valinnalla voidaan sisällyttää vaiheen alkuun tietuetyypin valinta. Valittavissa olevat tietuetyypit ja niiden käyttämät kenttäjoukot asetetaan Recordtype Field Set Map -kentässä. Yksittäinen tietuetyyppi ja sen kanssa käytettävä kenttäjoukkomääritelmä kirjoitetaan muodossa: tietuetyypin Api-nimi, kaksoispiste ja

viimeisenä kenttäjoukon Api-nimi. Määritelmät erotellaan toisistaan pilkulla.

Mukautetussa metadatassa on myös hakusuhdetyyppisille kentille lisäasetuksia, kuten kenttä, jonka perusteella haetaan ja uuden tietueen luomiseen käytettävä Lightning-komponentti.

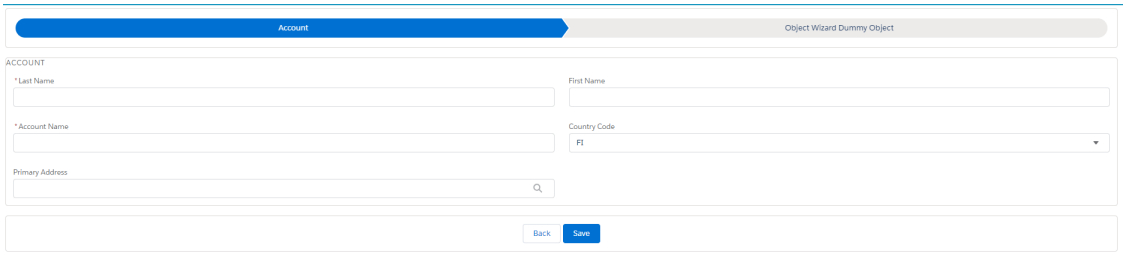
## 4.2 Lightning-komponentit

Sovelluksen käyttöliittymä on tehty Lightning-komponenteista, joissa hyödynnetään SLDS-tyylejä. Lopullinen toteutus perustuu moneen erilliseen komponenttiin ja niiden väliseen kommunikaatioon. Erilliset komponentit tehtiin uudelleenkäytettävyys ja geneerisyys isoimmassa roolissa. Komponentit käyttävät Lightning-tapahtumia viestintään keskenään ja erilaisia tapahtumia on monia.

Kaikki tässä kappaleessa kuvatut Lightning-komponentit ovat minun tekemiäni. Toteutuksessa hyödynnetään myös Appiphonyn Strike-komponentteja kehityksen nopeuttamisen takia. Strike-komponentit ovat valmiita Lightning-komponentteja, jotka ovat vapaasti ladattavissa ja Salesforce -organisaatioihin asennattavissa BSD 3-lisenssin puitteissa.

### 4.2.1 OrderOnePager

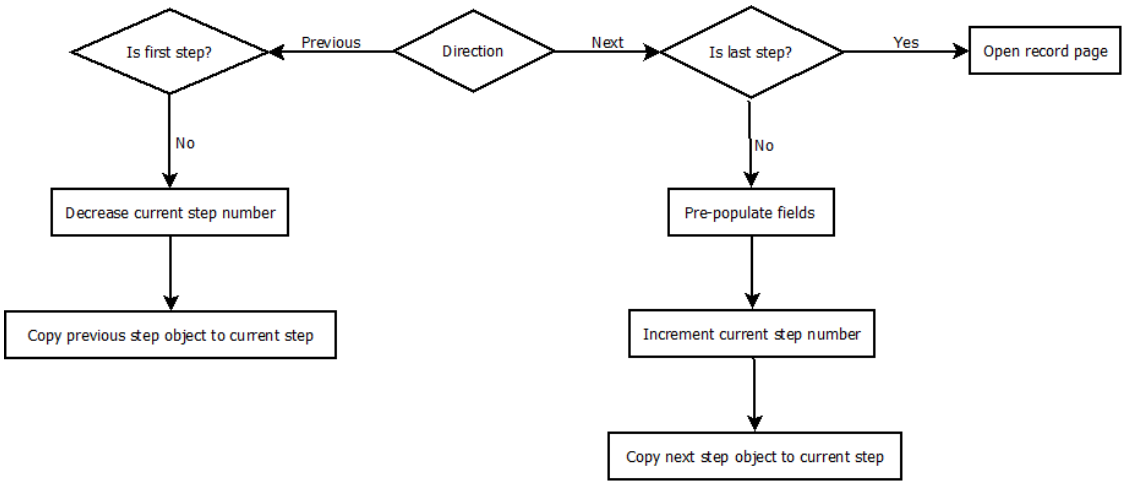
OrderOnePager-komponentti sisältää kaikki muut komponentit. Tämä komponentti hoitaa ohjelman alustuksen sen attribuuttiin asetetun asetuksen perusteella, eli muokatun metatiedon, johon asetukset on määritelty. Komponentti tekee kutsun back-end-osioon ja muodostaa sen attribuutteihin kaikki ohjelman vaiheet talteen yhdellä kerralla. Kutsu sisältää vain muokatun metatiedon rajapintanimen ja sen vastauksena tulee JSON-merkkijono, joka muunnetaan taulukoksi JavaScript-objekteja. Taulukosta yksi alkio sisältää yhden vaiheen objektin tarvittavat metatiedot. Nykyinen vaihe pidetään tallessa erillisessä attribuutissa, ja muut komponentit käyttävät viitettä siihen käsitellessään sen arvoja.




Kuva 3: OrderOnePager-komponentin ulkonäkö

Kuvassa 3 näkyvän komponentin asettelun yläossa on ObjectWizardProgerssIndicator-komponentti, keskellä ObjectCreator-komponentti ja alhaalla ObjectWizardButtons-komponentti. Myös ObjectWizardErrorNotification- ja ObjectWizardLoadingOverlay-komponentit ovat tämän komponentin näkymässä, oletusarvoisesti piilotettuina. Jos nykyisen vaiheen asetuksissa on määritetty tietuetyypin valinta, näytetään ObjectCreator-komponentin sijasta ObjectWizardRecordTypeSelect-komponentti, kunnes tietuetyyppi on valittu.

Kenttien arvojen esitäyttö hallitaan tässä komponentissa niiden kenttien osilta, joissa on määritelty lähdearvoksi jonkun toisen vaiheen objektin kenttä, joko suoraan tai osana kaavaa. Esitäyttö suoritetaan siirrittäessä uuteen vaiheeseen ja koskee vain vaihetta, johon siirrytään. Jokainen vaiheelle tehty kentän esitäyttöasetus käydään yksitellen läpi ja arvot asetetaan attribuutissa tallessa olevan objektitaulukon alkion kenttäolioiden arvomuuttujaan, jonka rajapintanimi vastaa asetuksessa olevaa kohdekenttää. Arvot kopioidaan samasta taulukosta, siltä objektioliolta ja sen kentältä, jotka täsmäävät asetusta. Lähdearvon ollessa kaava sijoitetaan lähdekentän arvo kohdekentän arvoon $-merkin tilalle.

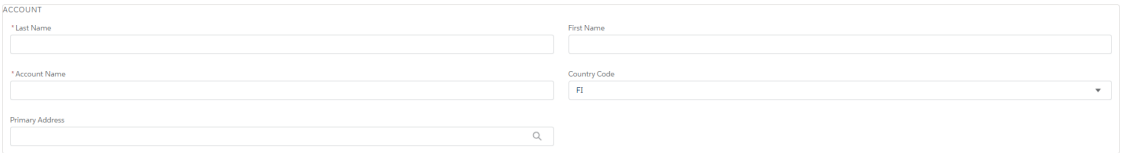



Kuva 4: Navigaatiotapahtuman aiheuttama tapahtumakulku

Komponentti kuuntelee ObjectWizardNavigationEvent-komponenttitapahtumaa, jonka tapahtuessa komponentti tarkistaa sen suunnan ja laskee sen mukaan vaiheen numeron, johon siirrytään. Seuraavaan vaiheeseen siirryttäessä kopioidaan koko olio nykyinen objektiattribuutista kaikkien vaiheiden taulukkoon siihen kohtaan, johon nykyisen vaiheen järjestysnumero viittaa. Seuraavan vaiheen esitäyttö suoritetaan ja nykyiseen objektiattribuuttiin kopioidaan kyseinen objektiolio, kaikkien vaiheiden taulukosta. Kun seuraavan vaiheen järjestysluku olisi isompi kuin vaihetaulukon koko, siirrytään viimeisen vaiheen vakiosivulle. Tämä tapahtumakulku on kuvattu kuvassa 4.

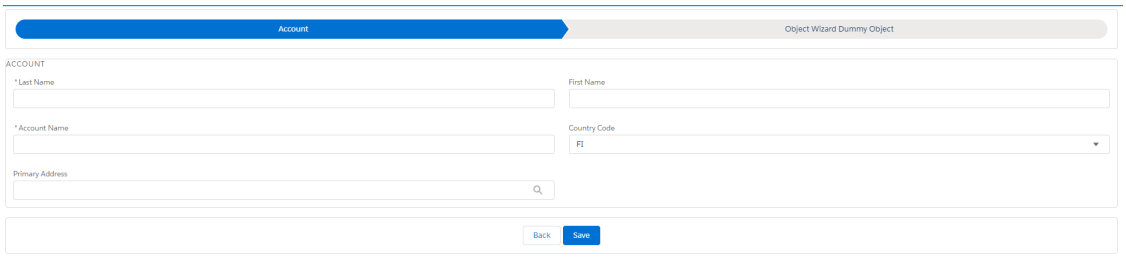
### 4.2.2 ObjectCreator

ObjectCreator-komponentti sisältää vaiheeseen liittyvät InputField-komponentit ja hoitaa kenttien arvojen tallentamisen. Komponentti luo kaikki sisältämänsä InputField-komponentit dynaamisesti sille parametrinä annetun objektitietueen perusteella. Hyödyntäen aura:iteration-elementtiä tämä tapahtuu kaikki komponentin näkymä- osiossa.




Kuva 5: ObjectCreator-komponentin ulkonäkö

Yksittäiset InputField-komponentit ovat tämän komponentin asettelussa jaettu allekkain kahteen sarakkeeseen kuvan 5 mukaisesti. Asettelu on toteutettu lightning:layout- ja sen sisällä lightning:layoutItem- standardikomponenteilla. Näkymän vasemmassa ylilaidassa kerrotaan nykyisen vaiheen objektin nimi ja koko komponentti on slds-card-elementin sisällä.

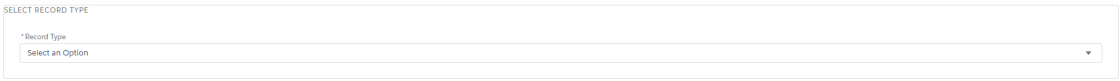



Kuva 6: Tallennustapahtuman käsittely

Komponentti kuuntelee ObjectWizardSaveEvent-applikaatiotapahtumaa ja reagoi siihen käynnistämällä kuvan 6 mukaisen tapahtumakulun, jossa aluksi käydään läpi jokainen komponentin sisältämä InputField-komponentti ja tarkistetaan, että kaikki tallentamiseen vaaditut kentät on täytetty. Jos vaadittu kenttä ei ole täytetty, asetetaan se virhetilaan. Kun kaikki vaaditut kentät on tarkistettu ja todettu täytetyiksi, komponentti kutsuu back-end-osion tallennusmetodia argumenttina koko objektitietue, joka tullaan tallentamaan. Jos tallennus onnistuu, tallennetaan vastauksena tuleva objektitietue attribuuttiin, jossa säilytetään nykysen vaiheen objektia. Onnistuneen tallennuskutsun lopuksi komponentti laukaisee ObjectWizardNavigationEvent-komponenttitapahtuman, jonka parametriksi on asetettu suunnaksi seuraava. Tallentaessa virhetilanteen sattuessa tämä komponentti laukaisee ObjectWizardErrorEvent-applikaatiotapahtuman, jonka parametrina on back-end-kutsun vastauksena saatu virheviesti.

### 4.2.3 ObjectWizardRecordTypeSelector

Jos asetuksissa on kyseiselle vaiheelle määritetty, että tietue-tyyppi tulee valita ennen muun tiedon syöttöä, näyteteään ObjectWizardRecordTypeSelector-komponentti.




Kuva 7: ObjectWizardRecordTypeSelector-komponentin ulkonäkö

Komponentissä on kuvan 7 mukaisesti yksi InputField-komponentti, joka on asetettu tietuetyypin valinta tilaan ja jolle on annettu parametrina valittavissa olevat arvot. InputField-komponenttia ympäröi slds-card-elementti.

### 4.2.4 InputField

InputField-komponentti näyttää yksittäisen kentän objektilta Salesforcen metadatan mukaisesti. Komponentille annetaan attribuutteihin objektin rajapintanimi, kentän rajapintanimi ja viittaus isäntäkomponentin attribuuttiin, johon kentän arvo tulee tallentaa. Komponentti hakee itsenäisesti kyseisen objektin kyseisen kentän tiedot ja mukauttaa näkymänsä sen mukaan. Esimerkiksi jos kenttä on tyypiltään valintaluettelo (eng. Picklist), näyttää komponentti listan mahdollisista arvoista, ja jos kenttä on tyypiltään hakusuhde, näyttää komponentti hakukentän kyseisiin objekteihin.




Kuva 8: Yksittäinen InputField-komponentti

Strike-komponentteja on tässä komponentissa hyödynnetty erityyppisien kenttien näyttämisessä, sillä ne ovat valmiiksi oikean näköisiä ja tarjoavat kaikki tarvittavat ominaisuudet. Erityisesti hakusuhdetyyppisen kentän toteuttamiseen olisi kulunut huomattavasti aikaa täysin itse tehtynä, sillä Salesforce ei ainakaan työn tekohetkellä tarjonnut vastaavaa vakiokomponenttia, ja erityisesti haun optimointi on osoittautunut hyvin haasteelliseksi aikaisemmissa kokeiluissa.

Myös kenttätason oikeudet otetaan huomioon, jos käyttäjällä ei ole oikeuksia muokata eikä nähdä kyseistä kenttää, jätetään kenttä näkyvistä. Vastaavasti jos käyttäjällä on pelkkä lukuoikeus kenttään, näytetään sen vain lukutilassa. Kuvassa 8 oleva InputField-komponentti on normaalitilassa.


Kuva 9: Komponentti virhetilassa

Tämä komponentti voidaan laittaa virhetilaan asettamalla sen hasError-attribuutin arvon todeksi. Tällöin kentän ympärille punaisen reunuksen ja näyttää virhetekstin kuten kuvassa 9 näkyy.

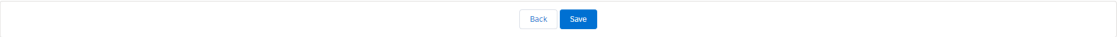
Arvojen muuttuessa komponentti ilmoittaa siitä muille komponenteille applikaatiotason tapahtumalla. Oletusarvoisesti InputField-komponentit kaikki kuuntelevat kyseistä ilmoitusta, mutta reagoivat siihen vain, jos kyseinen komponentti on määritelty olemaan riippuvainen ilmoittavan komponentin kentän arvosta. Tapahtuman parametrina on kentän uusi arvo, vastaanottavalla komponentilla on tallessa kyseiselle arvolle valittavissa olevat ali-arvot, ja se asettaa ne valittaviksi näkymän listaan. Jos vastaanottavalla komponentilla ei ole yhtäkään näkyvää arvoa kyseiselle arvolle, niin komponentti piilottaa itsensä. Tämänlaisen totetuksen ansiosta voidaan simuloida riippuvaisia valintaluetteloita (eng. Dependent Picklist).

InputField-komponentti on vahvasti riippuvainen sen back-end-ohjainluokasta, sillä Salesforcen kenttien metadataa ei voi käyttöliittymäkoodissa hakea. Täysin oman back-end-luokkansa ansiosta voidaan komponenttia uudelleenkäyttää missä vain komponentissa, kun halutaan näyttää jokin objektin kenttä.

Kutsussa back-end osioon siirretään objektin nimi, kentän nimi ja alkuperäinen arvo, joka on määritelty esitäytön aikana. Vastauksena back-end-kutsusta tulee kentän tarkat metatiedot, joiden perusteella komponentti muokkaa näkymänsä. Komponentin näkymä rakentuu aura:if-ehtolauseiden perusteella, joissa tarkistellaan kentän tyyppiä.

### 4.2.5 ObjectWizardButtons

ObjectWizardButtons on yksinkertainen komponentti, joka sisältää nappulat tallentamiseen ja takaisin siirtymiseen. Komponentti ei itsessään nappulan painalluksessa tee mitään muuta kuin laukaisee nappulan toimintoon liittyvän tapahtuman. Komponentti ei tiedä ohjelman tilanteesta mitään, sillä on vain kaksi tapahtumaa rekisteröitynä. ObjectWizardNavigationEvent-komponenttitapahtuma laukaistaan takaisin siirtymisnappulaa painettaessa, viestillä ”previous”. Tallennusnappia painettaessa laukaistaan ObjectWizardSaveEvent-aplikaatiotapahtuma ilman mitään parametreja.




Kuva 10: ObjectWizardButtons-komponentin ulkonäkö

Komponentin nappulat ovat Lightning Component Frameworkin standardeja lightning:button-komponentteja, jotka noudattavat jo valmiiksi SLDS-tyylitystä. Nappuloiden korostaminen on tehty käyttäen nappulakomponentin variant-parametria, johon asettettiin joko arvo ”neutral” tai ”brand”, SLDS-määritysten mukaisesti. Kuvassa 10 näkyy komponentin asettelu.

### 4.2.6 ObjectWizardProgressIndicator

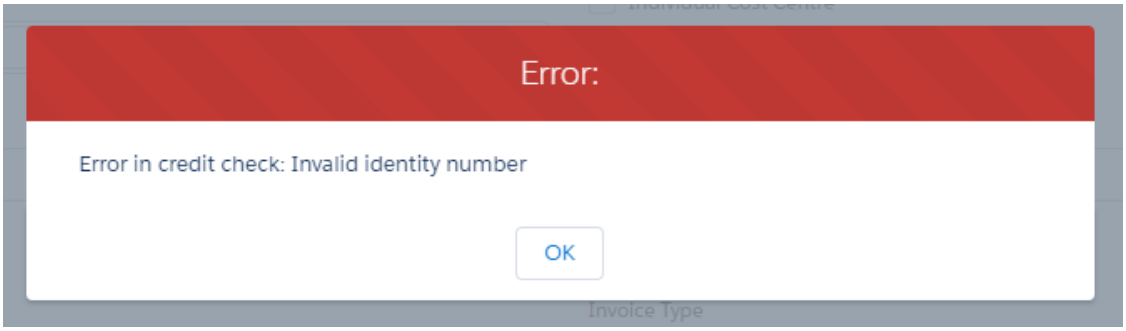
ObjectWizardProgressIndicator-komponentti sisältää yhden strike_wizard-komponentin, joka näyttää vaiheet ja niiden tilan. Parametreina tämä komponentti ottaa vaiheiden nimet, nykysen vaiheen numeron ja valmiiden vaiheiden lukumäärän. Komponentti hoitaa vaiheen valmistumisen nextStep-nimisellä metodilla, joka vain kutsuu lapsikomponentin advanceProgress- metodia.


Kuva 11: ObjectWizardProgressIndicator-komponentin ulkonäkö

Kuvassa 11 on esimerkki komponentin asettelusta.

### 4.2.7 ObjectWizardErrorNotification

ObjectWizardErrorNotification-komponenttia käytetään mahdollisissa virhetilanteissa näyttämään käyttäjälle virheviesti. Komponentti on aina taustalla, mutta ei näytä mitään ellei sen sisäinen näkyvyys- attribuutti ole asetettu todeksi. Komponentti kuuntelee ObjectWizardErrorEvent-applikaatiotapahtumaa, jonka tapahtuessa se asettaa näytettävän virheviestin tekstin ja modaalin otsikon, tapahtuman paramitrien mukaisesti. Komponentti asettaa itsensä näkyväksi vastaanottessaan tapahtuman ja pois näkyvistä, kun sen sulkee.

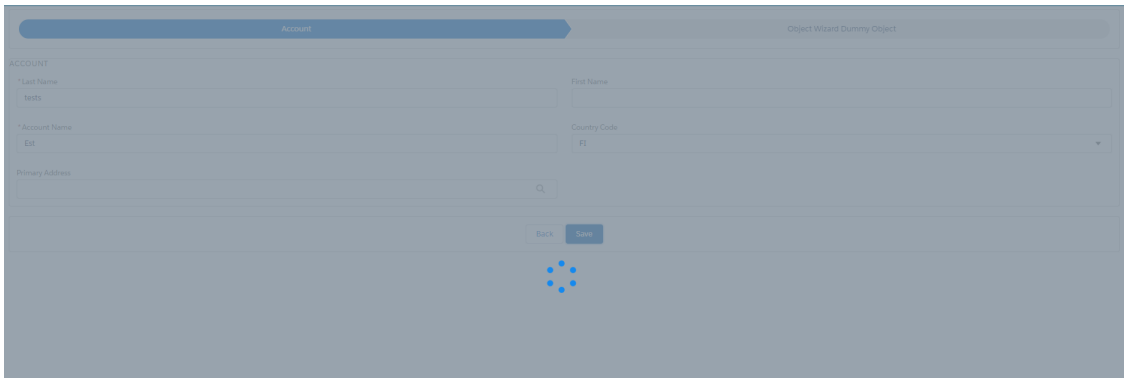



Kuva 12: ObjectWizardErrorNotification-komponentin ulkonäkö

Kuvassa 12 näkyy, että ulkonäöltään komponentti on yksikertainen moodaaliikkuna, jossa kerrotaan tietoja tapahtuneesta virheestä. OK- nappi sulkee ikkunan.

### 4.2.8 ObjectWizardLoadingOverlay

ObjectWizardLoadingOverlay-komponenttia käytetään tallennuksen aikana kertomaan käyttäjälle, että muutoksia ei voi enää tehdä. Komponentti kuuntelee ObjectWizardLoadingEvent-applikaatiotapahtumaa, jonka parametrin mukaan se asettaa itsensä näkyväksi.




Kuva 13: ObjectWizardLoadingOverlay-komponentin ulkonäkö

Tämä komponentti sisältää koko sivun kokoisen tummennetun elementin, joka tulee kaiken muun päälle ja estää interaktion alle jäävän kanssa. Komponentin keskellä on animoitu SLDS-ohjeistuksen mukainen latausindikaattori kuten kuvassa 13 näkyy.

### 4.2.9 ObjectWizardModalContainer

ObjectWizardModalContainer on komponentti, jota käytetään, kun tarvitaan jokin komponentti näkyväksi erillisen modaali-ikkunan sisällä. Käytetään hakusuhdekenttien uuden tietueen luonnin yhteydessä, jos asetuksissa on määritelty. Komponentti kuuntelee ObjectWizardNewLookupRecordEvent-applikaatiotapahtumaa jonka parametrinä tulee komponentin nimi, joka tulee näyttää. Modaalin sisään tuleva komponentti instantioidaan dynaamisesti $A.createComponent-funktiolla. Luotavalla komponentilla on oltava ulospäin näkyvät onSave- ja onCancel- Aura.Action-tyyppiset attribuutit, jotka tämä komponentti ylikirjoittaa omillaan. On oltava myös metodi tallennukselle, joka ottaa argumenttina callback-funktion suoritettavaksi muun logiikan jälkeen.

## 4.3 Lightning Out Visualforce-sivulla

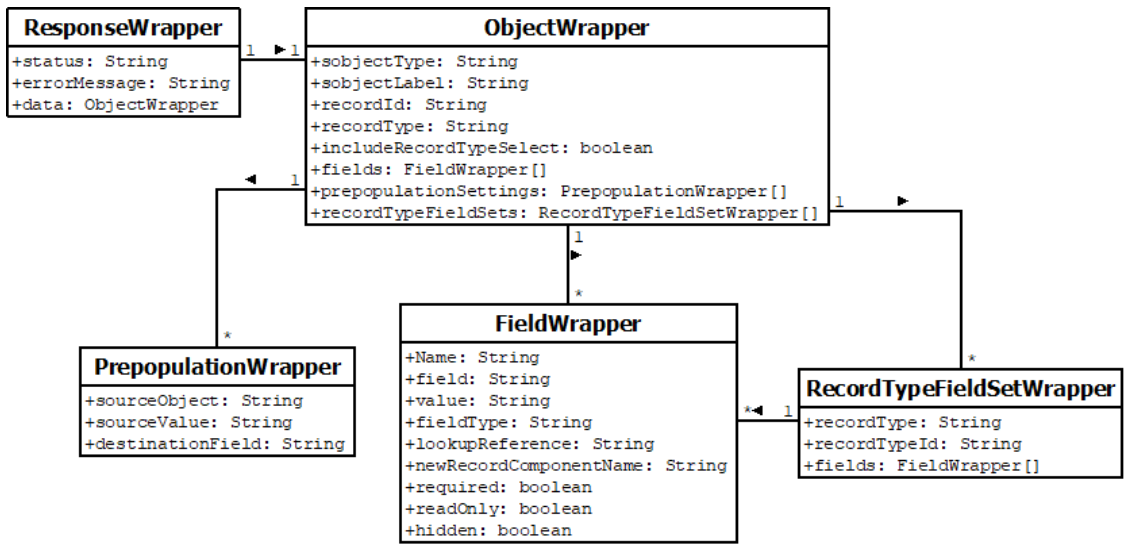
Jotta sovellus saataisiin käytettäväksi myös Classic -näkymässä ilman, että käyttöliittymä tehtäisiin kokonaan uusiksi, hyödynnettiin työssä Salesforcen tarjoamaa Lightning out-toiminnallisuutta Visualforce-sivulla. Sovellukselle tehtiin oma mukautettu välilehti, jossa on tämä Visualforce-sivu upotettuna. Visualforce-sivu välilehdet ovat käytettävissä myös

Lightning Experience -näkymässä, joten sovelluksen käyttökokemus pysyy samana kummassakin näkymässä, vaikkakin sovelluksen ulkonäkö ei vastaa Classic -näkymää.

## 4.4 Apex back-end

Palvelinkoodin kirjoitin Apexilla, se sisältää metodeita objektien tietojen hakuun ja tallentamiseen. Back-end-koodi käsittelee sovelluksen asetukset ja valittujen objektien metatiedot ja muotoilee ne käyttöliittymäkoodille sopivampaan muotoon.

Sovelluksen kaikki komponentit InputField-komponenttia lukuunottamatta hyödyntävät yhtä ja samaa Apex-ohjainluokkaa. Sovelluksessa on pyritty hoitamaan mahdollisimman paljon logiikasta käyttöliittymän puolella. Joitakin toiminnallisuuksia ei tosin nähty mahdolliseksi tai järkeväksi käyttöliittymä-koodissa suoritettavaksi, esimerkiksi tiedon tallentaminen ja metatietojen haku olivat tämänlaisia asioita.
Kaikki logiikka, joka ei ole yhteistä jokaisen vaiheen kanssa, on toteutettu back-end-luokassa, jotta komponentit olisivat mahdollisimman geneerisiä ja dynaamisia. Tili-objektin kanssa oli vaatimus jälkiprosessoinnille, joka koski vain sitä. Muun muassa henkilötilitietueeksi muuntaminen ja olemassa olevan tilitietueen päivittäminen syötetyllä tiedolla, on kokonaan toteutettu back-end-koodissa.



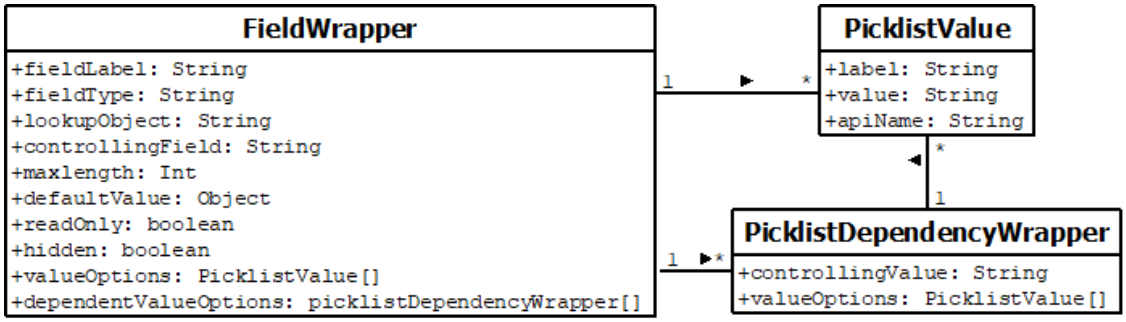

Kuva 14: Object Wizard-kääreluokkien kuvaus

Ohjelman alustuskutsu ottaa argumenttina vastaan mukautetun metatiedon nimen,

hakee sen kaikki tietueet järjestyslukujärjestyksessä SOQL-kutsulla ja muodostaa niistä ohjelman kaikki vaiheet. Yhden vaiheen muodostustamisessa aluksi muodostetaan kuvan 14 mukainen ObjectWrapper-olio, jolle asetetaan tyyppi ja nimi asetuksen mukaan. Seuraavaksi käydään läpi hakusuhdekenttiin liittyvät asetukset, joista muodostetaan avain-arvo-pareja, joiden avaimena toimii kentän nimi, käytettäväksi myöhemmin. Esitäyttö-asetuksista ensimmäisenä muodostetaan samoin avain-arvo-pareja niistä, joissa on kovakoodatut lähtöarvot. Muista luodaan PrePopulationWrapper- olioita sillä niiden arvot tullaan täyttämään lightning-komponenteissa.

Jos asetuksissa on merkitty tietuetyypin valinta, luetaan siihen liittyvät asetukset ja muodostetaan niistä RecordTypeFieldSetWrapper-olioita. Yhdelle oliolle tulee tietuetyypin nimi, tyypin tunniste ja lista kentistä. Kenttälista haetaan asetuksen mukaisesta kenttäjoukosta ja niiden esitäyttöön käytetään aikaisemmin luotuja esitäytön avain-arvo-pareja. Myös hakusuhdekenttien asetuksista muodostetut avain-arvo-parien arvot asetetaan oikeille kentille. Lopulta lista luotuja ObjectWrapper-olioita serialisoidaan JSON-merkkijonoksi.

Tietueiden tallennukseen käytettävä metodi ottaa vastaan yhden merkkijonon argumenttina ja muunnetaan takaisin ObjectWrapper-olioksi. Olion objektinimen perusteella luodaan uusi instanssi SObject-luokasta Schema-luokan avulla. Luodun SObject-olion kenttiin asetetaan arvot vastaanotetun ObjectWrapper-olion FieldWrapper-olioista ja se tallennetaan DML-operaatiolla tietokantaan. Tallennuksen onnistuessa palautetaan sama olio joka otettiin vastaan, sillä erolla, että sille on lisätty yksilötunnistekenttä. Vastaus muodostetaan ResponseWrapper-olioksi, jonka tila on ”onnistui” ja datana on kyseinen olio. Virhetilanteissa käytetään samaa kääreluokkaa, mutta sen tila ”epäonnistui” ja siinä on mukana virheviesti.




Kuva 15: Input Field -kääreluokkien kuvaus

InputField-komponentin back-end-ohjain sisältää vain yhden metodin, joka hakee Schema-luokkien avulla sille argumentteina annetun kentän metatiedot. Vastaus rakennetaan kääreluokkiin, jotka on määritelty tämän luokan sisään. Kuvan 15 mukaisen kääreluokan olio muunnetaan Apexin JSON.serialize-metodilla merkkijonoksi ja palautetaan metodin lopussa. Kääreluokat sisältävät muuttujat jokaiselle tarvittavalle metatiedolle ja niiden rakenne on pitkälti sama ohjelman alustuksessa käytettyjen kääreluokkien kanssa.

# 5 Johtopäätökset

Dynaamisen koodin tekeminen osoittautui aikaa vieväksi prosessiksi, vaikkakin Apex-kielen Schema-luokat olivat suhteellisen helppokäyttöisiä ja hyvin dokumentoituja. Uusien vaatimuksien toteuttaminen monimutkaisti konfiguroitavissa olevia asetuksia, hyvänä esimerkkinä hakusuhdekenttiin liittyvät asetukset. Asetuksista ei välttämättä tullut niin helppokäyttöisiä kuin olisi voinut toivoa. Tilannetta voisi parantaa esimerkiksi tekemällä niiden tekemiseen oman käyttöliittymän.

Osan työssä tehdyistä komponenteista voisi vaihtaa Salesforcen tarjoamiin vastineisiin. Työn tekohetkellä näitä vastaavia komponentteja ei valmiina ollut saatavilla, joten on käytetty joko Strike-komponentteja tai tehty omia. InputField-komponenttia on työn jälkeen käytetty monessa muussa komponentissa, joten tavoitteet sen uudelleenkäytettävyydeestä täyttyivät, myöskin latausindikaattori- ja virheilmoitus- komponentteja on käytetty muuallakin. Itse kokonaisuutta ei ole vielä käytetty muuhun kuin tilauksiin liittyvissä asioissa, joihin se alunperin luotiinkin. Kokonaisuus olisi mahdollista paketoida ja tuotteistaa muiden asiakkaiden käyttöön. Tämä vaatisi back-end-koodista kaiken vaihekohtaisen koodin poistamista.

Lightning-tapahtumien osalta olisi voinut minimoida kokonaisuuden monimutkaisuutta käyttämällä geneerisempiä tapahtumia moneen eri tarkoitukseen sen sijaan, että jokaisella käyttötapauksella on määritelty oma tapahtuma. Loppujen lopuksi työ onnistui suhteellisen hyvin ja kokonaisuus toimii, siinä mihin se oli tarkoitettukin.

## Lähteet


1 Apex Developer Guide - Execution Governors and Limits... Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_gov_limits.htm>. Luettu 01.05.2018.

2 SOAP API Developer Guide - Object Basics... Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.api.meta/api/sforce_api_objects_concepts.htm>. Luettu 07.04.2018.

3 SALESFORCE HELP - Validation Rules... Verkkodokumentti. <https://help.salesforce.com/articleView?id=fields_about_field_validation.htm&type=5>. Luettu 01.05.2018.

4 Introducing custom metadata types: the app configuration engine for Force.com... Verkkodokumentti. <https://developer.salesforce.com/blogs/engineering/2015/04/custom-metadata-types-ga.html>. Luettu 06.04.2018.

5 SALESFORCE HELP - Custom Metadata Types... Verkkodokumentti. <https://help.salesforce.com/articleView?id=custommetadatatypes_about.htm&type=5>. Luettu 06.04.2018.

6 Apex Developer Guide - Schema Namespace... Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_namespace_Schema.htm>. Luettu 01.05.2018.

7 Apex Developer Guide - Describing sObjects Using Schema Method... Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_dynamic_describeSObject.htm>. Luettu 01.05.2018.

8 Apex Developer Guide - Data Manipulation Language... Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_dml.htm>. Luettu 01.05.2018.

9 Apex Developer Guide - DML Statements vs. Database Class Methods... Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_dml_database.htm>. Luettu 01.05.2018.

10 Lightning Components Developer Guide - What is the Lightning Component Framework? - Components... Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/intro_components.htm>. Luettu 30.03.2018.

11 Lightning Components Developer Guide - Controlling Access... Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-

us.lightning.meta/lightning/access_intro.htm>. Luettu 29.04.2018.

12 Lightning Components Developer Guide - What is the Lightning Component Framework? - Events... Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/intro_events.htm>. Luettu 30.03.2018.

13 Lightning Components Developer Guide - Component Handling Its Own Event... Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/events_component_handling_itself.htm>. Luettu 29.04.2018.

14 Apex Developer Guide... Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_classes_annotation_AuraEnabled.html>. Luettu 30.03.2018.

15 Lightning Design System - Getting Started... Verkkodokumentti. <https://www.lightningdesignsystem.com/getting-started/>. Luettu 30.03.2018.

16 Salesforce Object Query Language (SOQL)... Verkkodokumentti. <https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql.htm>. Luettu 30.03.2018.